%1 and others can also be distinguished with color/bold etc.

(ns syntax.test)

"A string is a sequence of characters"

#"regexp"

nil

(into [] empty-vector)

(into {} empty-map)

(into #{} empty-set)

(def var-name data)

(def+ var-name data)

(defn func-name [two-unnamed-vars \_ \_ & argn])

(defmacro macro-name [arg1 arg2 & argn])

(ƒ [arg1 arg2 & argn])

(ƒ+ [arg1 arg2 argn])

(deƒ+ func-name [arg1 arg2 argn])

(def-decorator :name [arg1 arg2 argn])

λ(shortcut-lambda %1 %2)

;; comparisons

(= 1 1)

(= 1 1)

(≥ 2 1)

(≤ 1 2)

;; booleans

((= 1 1) ⊤)

((= 1 2) ⊥)

((¬ (= 1 2)) ⊤)

;; threading

(-> f (inc))

(->> f (dec))

(↦ f t)

(mapv x f)

(vec f)

(vector x)

;; operations on functions

(! func) ;; complement

(∘ func1 func2) ;; composition

(& juxt1 juxt2) ;; juxtaposition

(\*> func param) ;; currying

(\*< func param)

(apply sum 1 2 3) ;; sum

(count f) ;; count

(java.lang.Integer/toString)

(.toString 1)

(ns syntax.test)

"A string is a sequence of characters"

#"regexp"

nil

(into [] empty-vector)

(into {} empty-map)

(into #{} empty-set)

(def var-name data)

(def+ var-name data)

(defn func-name [two-unnamed-vars \_ \_ & argn])

(defmacro macro-name [arg1 arg2 & argn])

(fn [arg1 arg2 & argn])

(fn+ [arg1 arg2 argn])

(defn+ func-name [arg1 arg2 argn])

(def-decorator :name [arg1 arg2 argn])

#(shortcut-lambda %1 %2)

(>= 2 1)

(<= 1 2)

(= 4 5)

(-> f (inc))

(->> f (dec))

(map f t)

(mapv x f)

(vec f)

(vector x)

(| func1 func2) ;; composition

(& juxt1 juxt2) ;; juxtaposition

(\*> func param) ;; currying

(\*< func param)

(apply sum 1 2 3) ;; sum

(count f) ;; count

(java.lang.Integer/toString)

(.toString 1)

(ns syntax.test)

"A string is a sequence of characters"

#"regexp"

nil

(into [] empty-vector)

(into {} empty-map)

(into #{} empty-set)

(def var-name data)

(def+ var-name data)

(defn func-name [two-unnamed-vars \_ \_ & argn])

(defmacro macro-name [arg1 arg2 & argn])

(fn [arg1 arg2 & argn])

(fn+ [arg1 arg2 argn])

(defn+ func-name [arg1 arg2 argn])

(def-decorator :name [arg1 arg2 argn])

#(shortcut-lambda %1 %2)

(>= 2 1)

(<= 1 2)

(= 4 5)

(-> f (inc))

(->> f (dec))

(map f t)

(mapv x f)

(vec f)

(vector x)

(| func1 func2) ;; composition

(& juxt1 juxt2) ;; juxtaposition

(\*> func param) ;; currying

(\*< func param)

(apply sum 1 2 3) ;; sum

(count f) ;; count

(java.lang.Integer/toString)

(.toString 1)